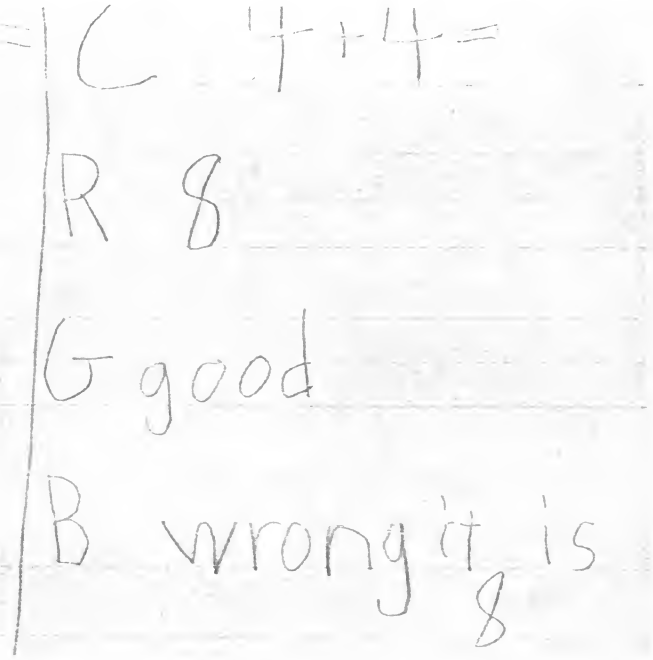
EDUCOM

Bulletin of the Interuniversity Communications Council (EDUCOM)

On the screen, an 8th grader has photographed by John Starkweather to be tested computer testing of his own design. At right: an arithmetic test written by a 1st grader from neighboring Don Tiburcio Elementary School.



Using a new "language" called Computest, California grade-schoolers write their own machine programs. They do it with

A Chatty Computer in Room B-3

TERRA LINDA'S VALLECITO INTERMEDIATE SCHOOL (450 pupils, 18 full-time teachers) could be the little red schoolhouse, *circa* 1966, recast in suburbia, and redrawn in Northern California Modern. Its walkways are covered, windows are wide, and the buildings are low and airy and rambling. But in one respect, in Room B-3, Valecito is unlike other schools.

Room B-3 is the province of an anthropomorphic IBM 1620 that refers to itself as "the friendly computer," and which 100 or more grade-school children engage each day in remarkably casual communication. After a few formalities, the machine often opens one of these dialogues by typing out: "Hello. I hope we will be able to have a good conversation. Please type your name the way you would like me to address you. When finished, press the key with R-S on it."

A small boy ponders the invitation on the electric typewriter that is the computer terminal. He leans over, hunts, and patiently pecks: "Tim R-S."

"I'm glad to meet you, Tim," replies the 1620. "I don't believe you have talked with me before." Within minutes, and here is the purpose behind the banter, Tim is responding on the keyboard to a test devised by his own classmates.

Machines are used increasingly to aid in instruction. But the experiment in the Dixie School District, of which Vallecito is part, on the hilly peninsula north of San Francisco's Golden Gate Bridge, differs in purpose. There, the object, as defined by Dr. John A. Starkweather, the University of California psychologist whose idea it was, is to foster "skill in questioning and problem solving through the programming efforts of the pupils."

Such examinations have ranged from reptiles and rocks (the latter programmed for its peers by a 6th Grade group) to fairly sophisticated interrogation on science, history, grammar, and the like. Typical was the following interaction between a 7th grader and the first program prepared by a classmate.

Computer: In ancient Rome, citizens were divided into what two classes?

Student: Senators and slaves.

Computer: You need to study harder. The two kinds of Roman citizens were patricians and plebeians. Try the next question . . .

The challenge to the children is the writing of questions that are straightforward and may be answered the same way. The key which opens the door to them to achieve this is a new computer language called Computest.

STARKWEATHER DEVISED COMPUTEST FOR USES QUITE different from those to which it is being put in the classroom. As director of the Computer Center and an associate professor of medical psychology at the UC San Francisco Medical Center, he envisioned the potentials of a coding language that would enable a person inexperi-

enced with computers, such as a physician, to arrange for a computer to communicate with someone in his natural language. What the researcher wanted, in other words, was a method by which computers could interact with humans in conversational fashion and, thus, assist in gathering information for medical diagnosis.

He began the work four years ago. During the same period, others have produced somewhat similar languages, tailored mainly to teaching, "computer-assisted instruction." Starkweather likes to call his approach "computer-assisted learning." He believes his to be unique in "the simplicity of its starting point, that is, how quickly one can get it to do something useful with very little information about it.

"Most computer programming techniques require a large amount of learning about a complete system before one obtains working results." He has always thought that "programming methods should be able to accomplish something analogous to the automatic transmission in the modern automobile. You shouldn't have to open the hood in order to get to where you want to go, unless along the way you get directly interested in what makes the machine run."

It was toward this minimizing of technicalities that he began building his own program, utilizing the Center's 1620, a relatively small device with a memory unit of 20,000 digits, indirect addressing, card-reader and punch, and console typewriter.

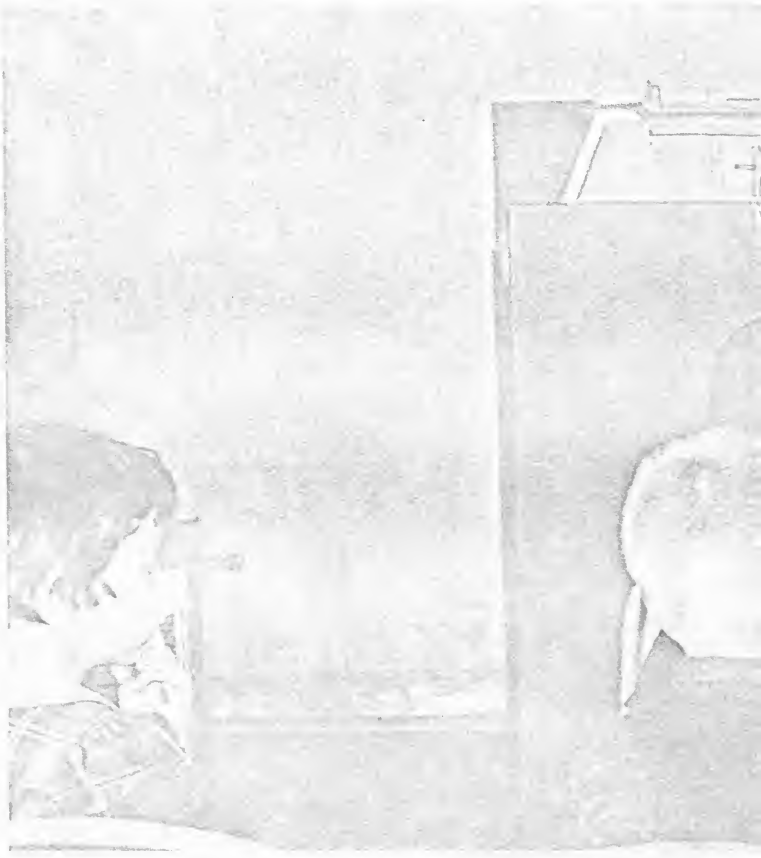
His aim, ultimately accomplished, was to program ways of instructing the machine to type statements or questions, and ways of storing recognition of true or false answers.

Other basic elements of the Computest language are the storage and typing of comments that depend on whether a reply is correct, and, not the least important, it allows a user a choice of strategies for evaluating an answer.

Statements and questions may be written in any form. Computest makes it possible to employ a variety of cues used to recognize replies. As Starkweather has said, "what the computer asks next is determined in part by the input it has just received from the typewriter."

Such a "sequential decision process" sets the approach apart from most computer diagnoses, which generally require considerable data before starting to make decisions. One psychiatrist, in his early experiments, simulated actual patients; it seemed to work well. There was one drawback: the machine was baffled by the responses of certain psychotics whose answers had nothing to do with the questions.

Almost by chance, meanwhile, a new route for inquiry appeared. To test the simplicity of his program, Stark-



John A. Starkweather

weather conducted a pilot trial with three eight-year-olds, his own 3rd Grade son among them. Given a brief introduction to the plan, they wrote in less than an hour a 12-question exam for classmates *(see print-out on page 6)* on arithmetic, science, and geography. "It seems to be a particular advantage in communicating with other children," the psychologist points out, "that questions are written in language of the age group."

The next move was obvious, since Starkweather also is an elected trustee of the Dixie School District (named, incidentally, by a settler from the South who erected the first school there a century ago). He talked about the success of his pilot trial with the district curriculum coordinator, Penrod Moss, and they decided to explore the possibilities further.

"The notion of the project then took on the form of the advantages to the kids to be derived from the actual practice of programming," Starkweather recalls.

"In order to write a sequence of questions, they would be forced to think of all the potential answers they would be willing to consider correct—and to predict the kinds of responses and terminology that someone taking such a test might use."

Although a cooperative research venture was agreed upon between the district and the University, finding

Abe Milstein explains to 4th grade pupils how they can prepare programs themselves by utilizing Computest language.

funds was more complicated. Some fund-granting agencies were less than eager to support it, at the outset. "They considered it an oddball project," Starkweather said. "First of all, our primary aim was not to produce instructional materials, which is the first thing one thinks of when he thinks of using a computer in this way. Any instructional materials that result will be purely a by-product."

Finally, the U.S. Office of Education decided to support the study for two years, effective last fall, through a contract to the University. Dr. Abe Milstein, a curriculum specialist, became program director in September. A month later, the Computer Center, which by then had installed a larger IBM 1401, moved the 1620 from San Francisco and into Room B-3.

Both Starkweather, as principal investigator, and Moss devote part time to the project; Milstein, a programmer, a punch-card operator, and a secretary work full time.

Starkweather has by no means abandoned the clinical applications for which he originally intended Computest. While he has not refined the language to a level satisfactory for patients, efforts continue: "I think it's bound to invade certain areas of information-gathering among patients. It already is possible, obviously, to machine-store and to deal with material gathered by questionnaires. It isn't difficult to imagine a patient typing answers to brief questions that would be presented either on a typewriter or via a small TV screen console."

He finds, even now, that Computest is valuable in training future physicians, simply because preparation of data for the computer requires precise knowledge of the subject. In addition, a program which plays the part of a patient can act as a laboratory exercise in interviewing.

"One project last summer," he explains, "was with a medical student who used this system to program an interview with a presumed patient complaining of a pain in the chest.

"I don't think the program itself was usable with patients by the time the summer program was over. But the student learned a lot about medicine in having to think through all the different problems that might be involved, and what kinds of questions to ask."

The point is, the Californian considers the Dixie Project and his Medical Center research separate prongs of the same tool. What he learns from one, then, will give him a firmer grip on the other.

A PROBLEM THAT ARISES WITH THE YOUNG PUPILS IS an almost inevitable ambiguity in questions. "At first, they're not precise enough," according to Starkweather. "The youngster discovers he's getting answers he didn't expect; yet they're valid enough depending on the interpretation of the question. When this occurs, the student begins to pinpoint his questions more precisely.

"One thing that I hope we will do is find out whether the children's questions are usually *who, where,* or *when* questions, as I think they will turn out to be. We shall then try to influence the children to write *why* questions, more open-ended questions.

This matter of question construction also intrigues a team of UC Berkeley psychologists who are analyzing creativity. Dr. Richard Crutchfield, who heads that investigation, has assigned a researcher to Dixie, as a "bridge" between the two efforts.

Predictably, when pupils are permitted to contrive programs along whatever lines they wish, these lines sometime follow the Beatles, the Rolling Stones, or Agent 007. But, perhaps paradoxically, the majority do not. They voluntarily pursue what educators term "academic content," topics, that is, closely tied to classwork, anatomy, authors, and astronomy, for example. Programs, indeed, have ranged pretty much the full span of gram-

```
PROGRAM NUMBER  103        PROGRAMMED BY
HELLO, I AM THE FRIENDLY COMPUTER.
WOULD YOU LIKE TO TALK TO ME.Q

TYPE YES OR NO, THEN PRESS THE KEY WITH R-S ON IT.

YES RS

GOOD.  I AM GLAD YOU UNDERSTAND WHAT I SAID.
I HAVE SOME QUESTIONS FOR YOU.

PLEASE TYPE YOUR NAME SO I WILL KNOW YOU.

MARK RS

I AM GLAD TO MEET YOU, MARK
HERE IS THE FIRST ONE.

WHO'S AN ELECTRIC GO THROUGH WOOD EASILY.Q

? RS

YOU SMARTY.  HOW DID YOU GUESS IT.

HOW LONG DOES A FLY LIVE.Q

1 DAY RS

YOU MUST BE SMART TO KNOW THAT.

WHAT IS A LEAPY FROG CALLED.Q

POLLYWOG RS

GOOD

WHAT IS A SHILLELAGH MADE OF.Q

I DON'T KNOW RS

YOU DONT COME FROM IRELAND.

TIMMY WANTS TO KNOW THE ANSWER TO THIS.
WHAT IS 7 X 9

67 RS   TYPE AGAIN .
63  RS

VERY GOOD, RING-A-DING.

IN BASEBALL, HOW MANY BALLS WALK THE BATTER.Q

4 RS

YOU MUST KNOW HOW TO PLAY BASEBALL.

WHAT CODE HAS DOTS AND DASHES.Q

MORSE RS

YOU GET THREE STARS ***

WHAT ARE YOUR TWO NEWEST STATES.Q

ALASKA HAWAII RS

YOU KNOW YOUR STATES.

WHERE DO KANGAROOS LIVE.Q

AUSTRALIA RS

GOOD

DO FATHER KANGAROOS HAVE POCKETS FOR BABIES.Q

NO  RS

YOU ARE A KANGAROO EXPERT.

WHAT KIND OF BIRD TALKS.Q

PARROT RS

HELLO, YOU FUNNY BIRD.
```

More about CHATTY COMPUTER

mar-school curricula. An important element of the Starkweather procedure is that the youngsters who have authored programs regularly receive print-outs of responses to their quizzes, allowing them to mend their methods, to expand and hone their inquiries. Similarly, other people who have taken the quizzes may have a chance to challenge authors.

"Was there a question in this program," one series asks, "that was incorrectly scored, or badly written, or confusing in meaning? If so, this is your chance to quarrel with the author. Type your objections. If not, press the R-S key."

Reversing the order of the familiar question-answer format, some programs let the subject ask the questions, as with this example.

Computer: You may ask me some of the main facts about the moon.

Student: Is there any life on the moon?

Computer: Probably not. Since there is no atmosphere on the moon, it is unlikely that living things exist there.

The 8th grader who programmed that one put in the answers for six specific points, and soon may increase the number. He is busily consulting sources.

THE PROJECT WAS BLUEPRINTED TO EXPLORE THE 4th through the 8th grades, although it encourages any pupil in the district who is interested. First graders have scrawled out simple math exams. By and large, the programs, once they are formulated, are card-punched by professionals. The author receives a print from these cards for review.

As the scheme functions, in general, Milstein first visits a classroom to explain Computest and, in broad terms, the machine. Programs are prepared, often on whatever subject is current in the class. Then the class, either as a whole or in small groups, is introduced to the computer itself. Each member sits at the typewriter to face some questions.

"Our first thought," says Starkweather, "was that teachers would learn this system, get confident with it, and begin to introduce it to the youngsters in their class.

"But we found this to be difficult, partly because it takes a while before a teacher gets comfortable enough with a new tool like this to be willing to pose as the expert before her class. So, rather than put her in that position, it seemed to be a little easier and more reasonable to go directly to the youngsters. The teacher learns it along with them and can feel comfortable about asking a question

Pilot-trial question series, which started work with children, was put together by group of three California eight-year-olds.

Eric Colby

In Vallecito classroom, Starkweather shows student how to use typewriter that is connected with an IBM 1620 computer.

or being in the same boat they were." This May, the Dixie Project is to add magnetic disc equipment that will give the computer conversations better branching (the departure by the machine from proceeding to the next instruction in line) and will multiply memory capacity 100 fold. And, by the end of the year, the project expects to use Computest from remote typewriter terminals connected to an IBM 360/40 at the San Francisco Medical Center.

But Starkweather feels a point already is being proved. He says: "If a machine is going to talk to you and you're supposed to talk back, as you would with a really active tutor, it ought to deal with you in your own language."

Apparently the pupils agree. One day, several of them were caught playing hookey from their regular class. They were in Room B-3, chatting with the computer.

Triangle Center

National Science Foundation funds are supporting what soon will be the largest educational computing complex in the nation, the Triangle Universities Computation Center near Durham. Duke University, the University of North Carolina, and North Carolina State University joined to establish TUCC. There are plans to expand its services by telephone lines, ultimately, to at least 76 other institutions in the state.

NSF announced that it would grant each of the three universities, which have about 33,000 students and faculty altogether, $500,000 (a total, that is, of $1.5 million) to help operate the facility. By August, the center will be completed with the installation of a leased IBM System 360/Model 75, a computer that can add 1 million 10-digit numbers in less than a second.

Telephone lines will tie it to System 360/30s on the three campuses, as well as to low-speed typewriter and medium-speed terminal devices installed at various places at the universities. A variety of disciplines, including medicine, engineering, and statistics will share time.

More Members

Twenty states now are represented in EDUCOM. Total institutional membership in the Council grew to 32 when the Board of Trustees, at a March meeting at Tulane University, approved applications from five additional schools. The new members are Indiana University, the University of Kansas, Notre Dame, Texas A & M, and Washington University in St. Louis.

Future Meetings

Apr. 26 - 28: SPRING JOINT COMPUTER CONFERENCE at Boston Sheraton Hotel, Boston. Contact: American Federation of Information Processing Societies, 211 East 43rd St., New York, N.Y. 10017.

May 10 - 12: NATIONAL TELEMETERING CONFERENCE at Boston Sheraton Hotel, Boston. Contact: IEEE, 345 East 47th St., New York, N.Y. 10017.

May 18 - 20: NATIONAL MEETING, OPERATIONS RESEARCH SOCIETY OF AMERICA, Los Angeles. Contact: Dr. John F. Walsh, System Development Corporation, 2500 Colorado Ave., Santa Monica, Calif. 90406.

June 15 - 17: COMMUNICATION CONFERENCE, INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS (IEEE), Sheraton Hotel, Philadelphia. Contact: Lewis Winner, 152 West 42nd St., New York, N.Y.

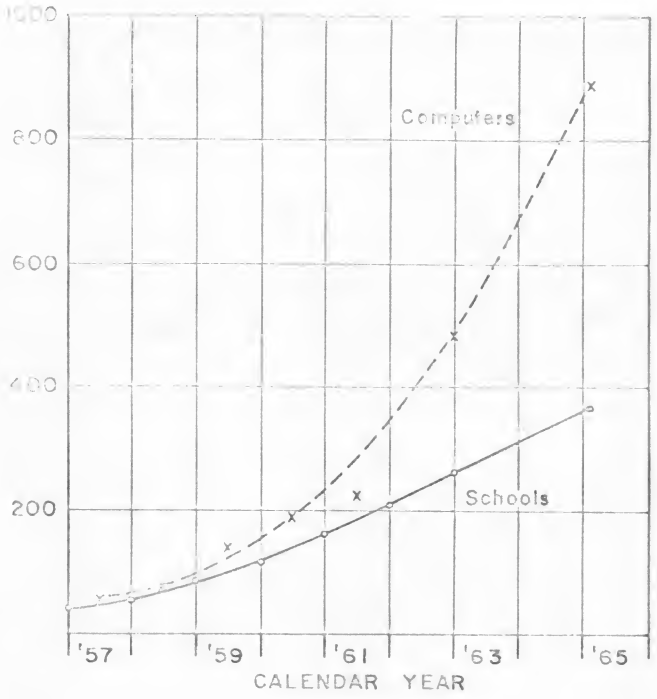
Nov. 8-10: FALL JOINT COMPUTER CONFERENCE at Civic Center, San Francisco. Contact: American Federation of Information Processing Societies, 211 East 43rd St., New York, N.Y. 10017.

March 1966

EDUCOM

Volume 1, Number 3

Bulletin of the Interuniversity Communications Council (EDUCOM)



Graph from NAS book shows schools with computers have increased but not so rapidly as total computers on campus.

DIGITAL COMPUTER NEEDS IN UNIVERSITIES AND COLLEGES

Universities and colleges in the United States should, by 1968, spend twice as much annually on computer facilities as they did in 1964. Their total computer capacities should have more than doubled. These suggestions come from a National Academy of Sciences ad hoc committee that completed more than a year of inquiry in December 1963. Unfortunately, it did not publish its findings for more than two years — until February 1966. By then, progress had easily exceeded panel predictions. Also, the group neglected, by and large, the role of the computer in language processing, including instruction and information storage and retrieval, and concedes that in general it "constrained to err on the side of conservative recommendations." Nonetheless, the report is significant.

IN 1957, THERE WERE ONLY 40 COMPUTERS ON AMERICAN campuses; there were almost 500 in 1964. But if the widening requirements of research and education are to

be met, an NAS Committee on Uses of Computers believes, then the rapid rate at which these machines have proliferated must continue. The Committee, chaired by Dr. J. Barkley Rosser of the Mathematics Research Center, University of Wisconsin, says that its study "shows a need for doubling of 1964 computing capacity by 1968, not only in regard to equipment but, more importantly, in regard to capacity for education and research in computer science.

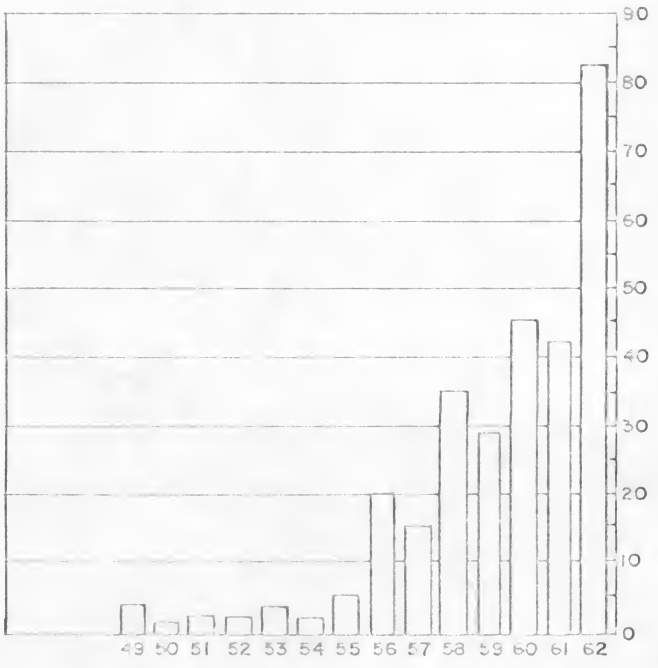
"Specifically, within the next three to four years, American universities and colleges (including the small colleges) will need help in acquiring (at a total cost of about $200 million) roughly 20 very large, 30 large, 600 medium, and 800 small [digital] computing installations."

That help, or much of it, it was proposed, must take the form of federal funds. The Rosser Report would have the institutional investment in such facilities swell to $528 million, by 1968, and the yearly budget for operating them rise to $300 million. By comparison, these figures in 1964 were, respectively, $250 million and $130 million.

The 1964 sums, the panel points out, represented "about 3 per cent of the education and research budgets of the universities, and [were] comparable with the cost of operating their libraries. Libraries, with their slow, predictable growth, have been financed by university funds, but the explosive growth of computing expense has been too much for the schools, and they have turned to the federal government for support. In 1954, federal support amounted to $65 million, which is about 5 per cent of the 1964 federal research and development expenditures at universities." (Federal financing of computers at universities totalled $87 million in 1965).

"The rate of growth of campus computing budgets has been twice as great as the growth of research on campuses," the Committee continues. "Thus the cost of operating campus computing centers has until recently been doubling every two years, at a time when campus expenditures for research and development have been doubling about every four years."

Number of schools setting up computing facilities (shown here by year) has risen rapidly. Total by 1965 was 801.



IN A LETTER OF TRANSMITTAL THAT ACCOMPANIES THE document, Dr. G. B. Kistiakowsky emphasizes that "new developments, which are apparent now but could not be forseen in detail at the time" the basic report was prepared "make it evident that the reality will and probably should depart upward from [its] projections."

He says that, "in particular, there is increasing evidence that needs for computer use in connection with formal undergraduate and graduate instruction were underestimated. Furthermore, computer education, especially at the undergraduate level, will have a heavy multiplying effect on the demand for computers in graduate education and basic research within a few years."

Kistiakowsky, the Harvard scholar who heads the Academy Committee on Science and Public Policy, also inserts a cautious note. He expresses fear that "an ever-expanding use of computers for the solution of scientific problems might change the nature of problems that active scientists choose for study, and thus change the whole nature of scientific research.

"We are concerned," he says, "lest this trend cause too many scientists to forget what a fabulously good computer the human brain is . . . We urge that, along with increased emphasis on computers in universities, there be an increased emphasis on education in classical applied mathematics."

IMPRESSIVE THOUGH THE PROJECTED EXPENDITURE increases may be, they would, in fact, grow at only approximately half the rate of similar increases over the past half dozen years. The NAS Committee, of which Dr. Thomas A. Keenan, director of the University of Rochester Computing Center, was executive director, puts the nationwide outlay for computers at $7 billion in 1964. Only about 4 per cent of the computers were located at insti-

(c o n t i n u e d o n p a g e 8)

tutions of higher learning. "Despite this small percentage," the report says, "colleges and universities have played a key role in computer development. The first computers were built at universities, and campus computing centers are currently developing more advanced systems programs that will permit all computer users easier and more satisfactory access to computers."

In addition to their use "by an increasing number of students," the digital devices are, "like laboratory equipment, needed to do research. They increase the effectiveness of other scientific equipment and permit many scientific studies of a scope and depth heretofore unattainable."

Therefore, the panel underlined a conviction "that channeling a larger fraction of expected increases in federal research and development will yield a considerably larger increase in the effectiveness of other scientific equipment, will greatly aid scientific workers, and will have important new applications elsewhere. Further, it will help train students to fill some of the 35,000 computer staff jobs being created yearly."

AT THE HEART OF THE ACADEMY REPORT* ARE FIVE somewhat broad recommendations, the first being the doubling of computer capacity, a goal that would require a rise in the federal share to something like $180 million in fiscal 1968. The other proposals:

(1) Development of "a number of strong campus research efforts" into regional centers. (2) Support "for novel, unforeseen research" as yet unsponsored. (3) Closer coordination among the eight or more federal agencies already aiding campus computation. And (4) planning for long-range cooperation between government and education *beyond* 1968.

Under the regional-center concept, central campuses might provide computing service for both their own and nearby schools, experimenting with various methods of doing so. According to the panel, "research groups at these centers, with several directions of emphasis, would play major roles in originating and developing new programming systems and languages (software), new ideas for auxiliary and remote equipment (peripheral hardware), unified planning of hardware and software (coordinated design), and particularly the advanced education of computer specialists. Equipment of specialized and advanced design must be made available to research groups of these kinds. Estimated costs for strengthening about half a dozen such efforts and purchasing advanced equipment should build up to a total of at least $10 million per year within the next four years above and beyond any costs of providing service computing."

Too little attention has been paid to training in the use of computers, the Committee contends, and it feels that "schools that develop a vigorous and imaginative program of education should not only have suitable financial help for the program itself, but might be given preferential treatment in other ways, say in allocation of funds for computers." Training in computer technology, put simply, may follow one of two directions: the teaching of students who use the machines merely for their own learning or for research, and the education of computer scientists. In either case, the panelists found, "support has lagged behind demand. Universities have greatly underestimated the need for training in connection with computers. They have lacked adequate support in this as well as in other areas of computation, but could probably have obtained more support had they stressed the need and presented a well-planned program."

Supplying trained personnel to run computers and to teach others how to run them is emphasized especially as "a point of considerable urgency." The report explains that "genuine sophistication in the use of computers has now developed and is increasing rapidly.

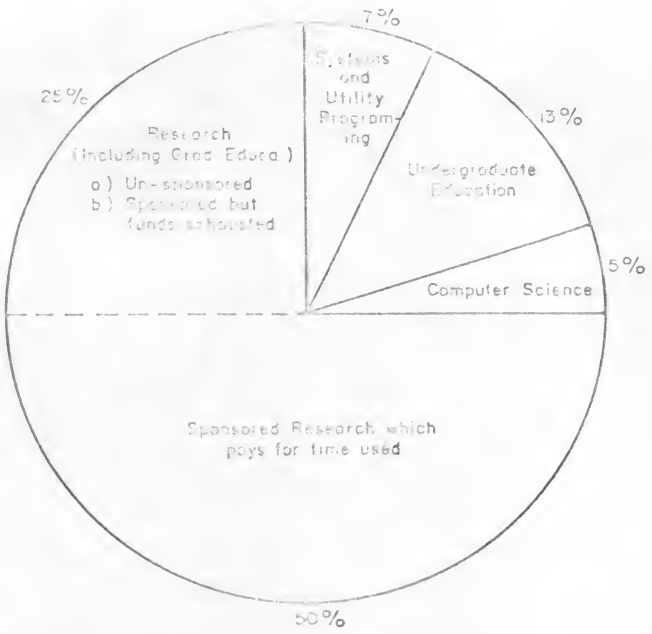
"Thus, operation of computers and instruction in their use by self-taught amateurs will be less and less acceptable, and the pressure for properly trained personnel for these and other matters connected with computers is

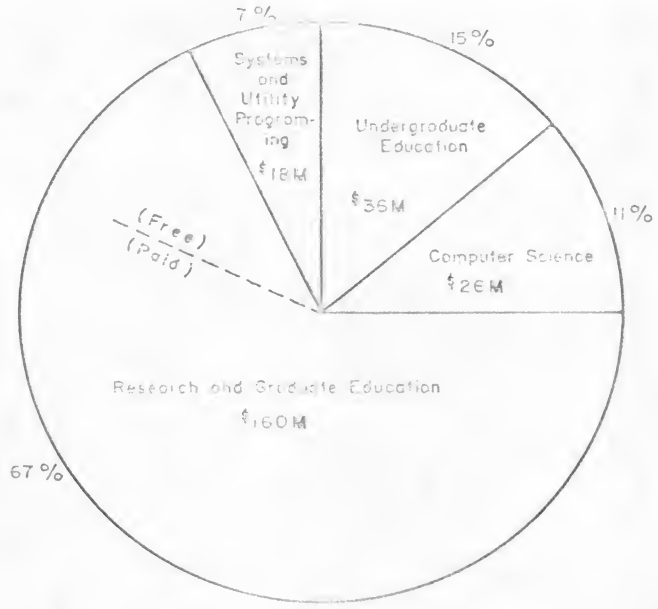
The EDUCOM Bulletin is being distributed, experimentally, among all faculty at member institutions. A survey is being prepared, however, to determine which faculty find it useful, and thereafter circulation of the publication will be directed to them.

Subscription rates are $5 annually to educators and students on non-EDUCOM campuses, and $10 to persons who are not educators.

Membership in EDUCOM is open to all accredited colleges and universities in the United States and Canada. Information may be obtained from the executive director, Dr. James G. Miller, Box 625, Ann Arbor, Michigan 48107.

* Besides Rosser and Keenan, Committee members were Hendrick W. Bode of Bell Telephone Laboratories; Gerald M. Clemence, Yale Observatory, W. J. Dixon, School of Medicine, and Arthur H. Rosenfeld, Department of Physics, both University of California; E. A. Matsen, Department of Chemistry and Physics, University of Texas; Philip H. Morse, Department of Physics, and Walter A. Rosenblith, Department of Electrical Engineering, both MIT; William R. Sears, Graduate School of Aeronautical Engineering, Cornell; Herbert A. Simon, Graduate School of Industrial Administration, Carnegie Tech; John W. Tukey, Department of Mathematics, Princeton; and James T. Wilson, Institute of Science and Technology, University of Michigan. Copies of the 176-page *Digital Computer Needs in Universities and Colleges* are available at $4.50 each from the National Academy of Sciences, 2101 Constitution Avenue, N.W., Washington, D.C. 20418.





Charts indicate how $100 million cost of campus computing was apportioned in 1965 (left), and how Committee thinks $240 million cost in year 1968 might be divided (at right).

growing sharply. Very few centers in the country now are capable of training professional staff in the modern and really efficient use of computers." As the Committee learned, "graduate education in computer science has already begun at a number of the major universities (such as Pennsylvania, MIT, Michigan, Stanford, Illinois, Wisconsin, Purdue, and others).

"However, the production of graduates from these programs at the doctoral level is as yet so small as to be negligible in comparison with the needs."

SOME OF THE STATISTICS THE NAS GROUP GLEANED ARE provocative. For instance, the birth of electronic computation did not occur until 1945 (when Mauchly and Eckert completed ENIAC), but, 20 years later, the combined calculating capacity of U.S. computers exceeded that of the nation's human populace by a factor of 10 or more.

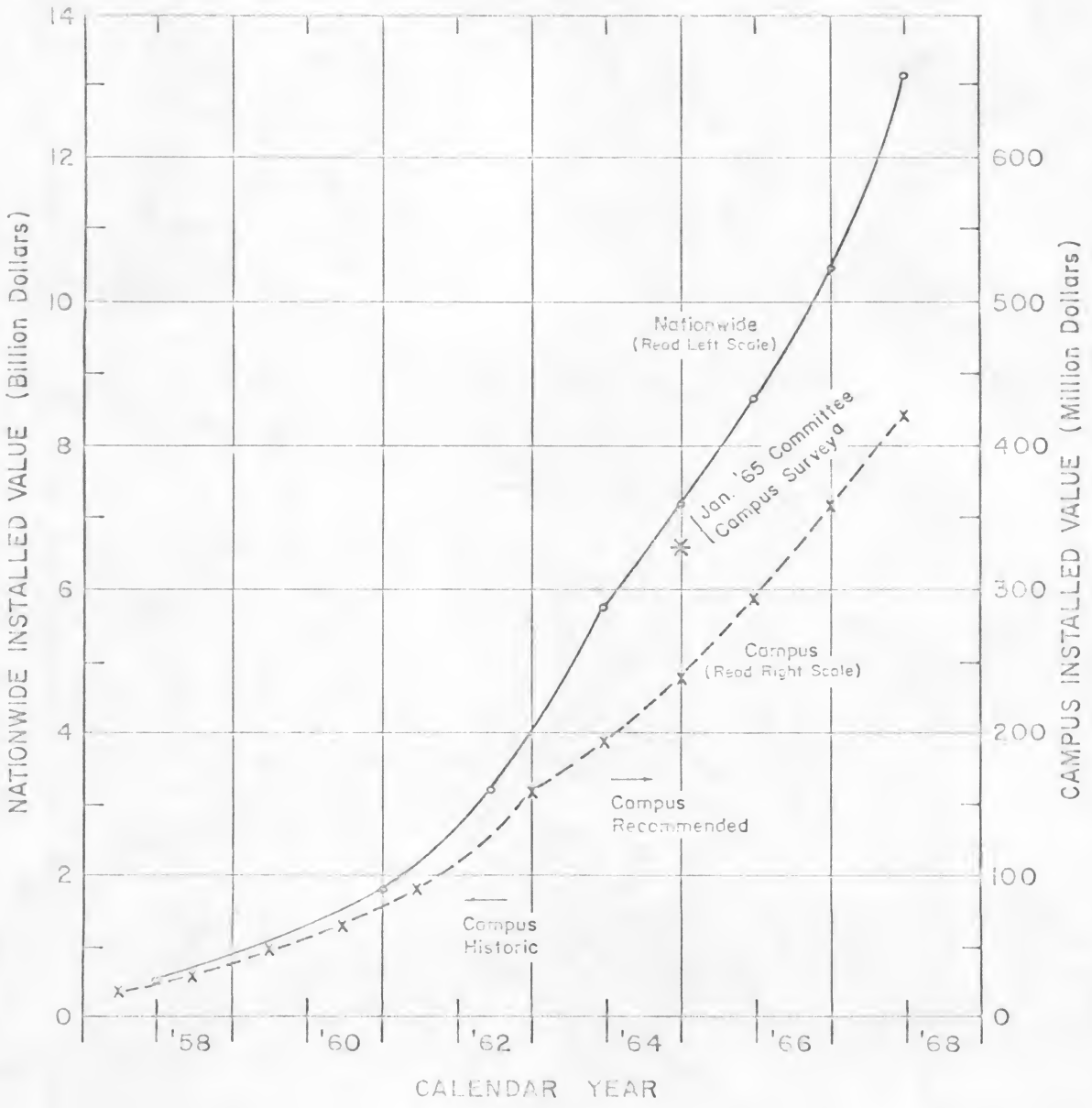
Computers worth in excess of $3.8 billion were shipped in 1964; conservative estimates are that 1970 shipments will total nearly twice that much. Although the aerospace industry accounted in 1964 for 5 per cent of the gross national product, the computer industry, without which it could not exist, accounted for less than half of 1 per cent of the GNP. Universities spent $6.6 million on computers in 1957, and $175 million in 1965. Even four years ago, 92 engineering schools (with 130,000 undergraduates) required the use of computers in at least some departments. In February 1965, there were 801 computers on campuses, but only 14 per cent of the colleges had one. California, with 96 campus computers, had the most, but Nevada led by percentage: one college, one computer.

The Academy chose the Committee on Uses of Computers from a variety of disciplines (the foreword says that "many members were initially not particularly informed about computers"). Consequently, the finished book reflects their interests. There are short discussions on the uses of, and needs for, computation in physics, astronomy, engineering, and the behavioral sciences, among other fields.

NAS panelists "expect that undergraduate courses that use computers in biology will be fairly common in five years, with perhaps half the undergraduate students having an opportunity to take such a course." The same was said for the full cross-section of curricula: "Computers will be of growing importance as an aid in teaching other subjects. Some of the potential uses lie in the correction and grading of assignments, the laboratory simulation of complex systems, and the computer's use as a teaching machine.

"Even the well-educated man," the Committee concluded, tends to think "of the computer system as a magical box, and of its use as incomprehensible . . . It is important to the social well-being of our country that the educated citizen understand computer science at least as well as he now understands medicine or mechanics."

Value of computers at universities, though mounting, still represents only 3-4 per cent of nationwide value (below).



Reprinted from

Comput. Biol. Med. Pergamon Press 1973, Vol. 3, pp. 293–298. Printed in Great Britain.

A Return to a Dedicated Machine for Computer-Assisted Instruction

MARTIN KAMP and JOHN A. STARKWEATHER

Office of Information Systems, University of California at San Francisco, San Francisco, California, U.S.A.

(*Received* 7 *May* 1973)

Abstract—Advances in computer technology have made it technically and economically feasable to produce a self-contained interactive computer terminal. A programming system called PILOT 73 will utilize the built-in minicomputer and carry out computer-assisted instructional programs with students. Comparisons are made between this dedicated system and the more traditional time-sharing multi-terminal computer system normally used for computerized instruction.

PERGAMON PRESS
OXFORD · NEW YORK

Comput. Biol. Med. Pergamon Press 1973, Vol. 3, pp. 293–298. Printed in Great Britain.

A Return to a Dedicated Machine for Computer-Assisted Instruction

MARTIN KAMP and JOHN A. STARKWEATHER

Office of Information Systems, University of California at San Francisco, San Francisco, California, U.S.A.

(*Received* 7 *May* 1973)

Abstract—Advances in computer technology have made it technically and economically feasable to produce a self-contained interactive computer terminal. A programming system called PILOT 73 will utilize the built-in minicomputer and carry out computer-assisted instructional programs with students. Comparisons are made between this dedicated system and the more traditional time-sharing multi-terminal computer system normally used for computerized instruction.

CAI Computer-assisted instruction Dedicated Minicomputer Terminal Language

INTRODUCTION

THE IDEA of using a computer for instruction originally grew from a combination of computer technology and principles of programmed learning as used in teaching machines. The first computer-assisted instructional systems, in the early 1960's, dedicated the resources of an entire computer to operate a single terminal station. This mode of operation was expensive, and served as a major deterrent to the early use of computerized instruction.

The advent of time-sharing systems eased the financial restrictions on the use of the computer for teaching. Such systems allow the computer to carry out several jobs during the same time period by allocating small divisions of the total time to each job in turn. In this way, a large number of interactive terminals can be operated simultaneously by a single computer. This significantly reduced the cost per student hour, and made possible the development of computer-assisted instruction as it exists today.

Continuing advances in computer technology may cause a return to a mode of operation in which there is once again a dedicated computer for each interactive terminal. The "smart" computer terminal, also called a "stand-alone" computer terminal, is now being produced in versions that are compact, powerful and competitively priced. Typically these devices consist of a cathode-ray tube display, a keyboard and a built-in mini-computer. They are capable of carrying out a variety of interactive programs without being connected to a larger time-sharing computer system.

THE COMPUTEST SYSTEM

Computer-assisted instruction has been in existence on the San Francisco campus of the University of California for more than 10 yr. It began with an interactive computer system called COMPUTEST[1] in 1962. This system ran on a small IBM 1620 computer and was able to carry out interactive programs with one user at a time via the console typewriter.

In early versions the COMPUTEST system operated with program material stored in a deck of punched cards, and read into the computer at the time of execution. Later, more sophisticated versions of the system had vastly increased capabilities because of moving-head disk storage units, but in common with many of the early CAI systems, COMPUTEST dedicated one computer to interact with one student.

THE PILOT SYSTEM

When an IBM 360 model 50 computer with time-sharing capability became available on the campus, a new computer-assisted instructional system called PILOT[2,3] was developed. PILOT stands for Programmed Inquiry, Learning Or Teaching, and the system is designed to simultaneously operate multiple remote terminals. The language features of the PILOT system were patterned after those of COMPUTEST, and stressed easy entry of instructional material by the program author. It was felt that a teacher should not have to become a computer expert in order to develop a computer-assisted instructional program. The PILOT system has been used for traditional frame-oriented instructional programs as well as providing practice with simulated clinical situations, self-evaluation testing, and simulated interviewing[4,5] with natural language input from the students.

INCREASING THE NUMBER OF USERS

All time-sharing systems have a limit on the number of terminal users that can be serviced at one time. The number of terminal access ports is a limitation on the total number of remote terminals that can be connected to a single computer installation, and each teleprocessing system operating on that computer has a practical upper limit of simultaneous users. The limit of each system is adjustable, depending on the computer resources that are allocated to it. Making a small increase in the number of users that can be handled by an interactive system may involve a large and costly increase in computer facilities such as a larger amount of core storage. In such a situation it may not be possible to justify the cost of the increased computer facilities that would be necessary on a large time-sharing system to permit a small increase in the capacity of the computerized instructional system.

The difficulty of increasing the terminal capacity of a large time-sharing system, as well as the availability of the latest "stand-alone" computer terminals, prompted the development of the system called "PILOT 73" that is described in this paper, and was written by John Starkweather. With an interactive instructional system operating on a stand-alone terminal device, increasing the number of users that can be served simultaneously is merely a matter of obtaining the desired number of stand-alone terminals, and does not depend on expensive and complex upgrading of a large time-sharing system.

EQUIPMENT

The CAI system described in this paper runs on a stand-alone computer terminal, the Datapoint 2200. This machine has a built-in minicomputer, two magnetic tape cassette drives, a standard ACSII keyboard and a cathode ray tube for character display. Random access semiconductor memory is used, and sizes from 4000 bytes (characters) to 48 000 bytes are available. The machine used for this project is equipped with 8000 bytes of memory. The display screen is 80 characters wide and has a capacity of 12 lines or 960 characters.

In operation, the stand-alone computer terminal is loaded with an operating system from a tape cassette. The PILOT 73 system, when loaded in this manner, goes into execution immediately and displays instructions for the student to load a program tape. Approximately half of the available memory is used for the PILOT 73 interpreter, and the remaining half is available for the code of the instructional program. About 100 000 characters of program code can be stored on one tape cassette and will be read into the main memory under control of statements written into the instructional program.

With a communications adapter and suitable programs, the stand-alone computer terminal can simulate a variety of conventional terminals. Programs are available to simulate an EBCDIC coded terminal (IBM 2741 equivalent), and an ASCII coded terminal at a variety of speeds. The ASCII simulator has the ability to read and write on the magnetic tape cassette. These simulators provide access to the facilities of large computers when needed, such as powerful text editors for program development or statistical programs for analysis of collected student data.

An assembly language and an assembler program to produce machine coded executable programs is available. The PILOT 73 system described in this paper was written in Datapoint assembly language. A higher level programming language and an editing system have also been provided. The editing system can be used by authors of instructional materials to enter the program code and create their own program on a tape cassette. The editor program has facilities for modifying programs already recorded on a tape, adding or inserting material and for copying from one tape to another. Since all users of this type of stand-alone terminal have the facility to duplicate tapes, the exchange of programs is greatly simplified.

THE PILOT 73 LANGUAGE

Shortly after the original version of the PILOT language was written for the IBM 360 computer, several versions of the language were programmed to run on other computers. Partly because of the requirements of different computer systems, and partly because of individual preferences, each version of the language was somewhat different from the others. In early 1973, representatives of six of the major versions of the PILOT language met together and agreed on a set of core language specifications to be common among all the systems. This language represents the accumulated experience of many CAI workers, and is called PILOT 73. Recognizing that differences will inevitably arise, a standard means of describing functions outside the core language has also been agreed upon. Because of the emphasis inherited from COMPUTEST and the original PILOT language, and because many of the systems are in elementary and secondary school settings, the result is a serious effort to design a CAI language that is easy for the program author to learn.

COMPARISON OF STAND-ALONE AND TIME-SHARING TERMINALS

In considering these two modes of implementing computer-assisted instruction, a number of comparisons can be made. The facilities available for program entry and data collection should be compared, as well as operational characteristics such as speed and reliability. Cost, ease of operation, expandability and suitability for certain types of instructional programs should also be explored. These must be considered to be overall comparisons, since specific characteristics of some systems, either stand-alone or time-sharing, may make them different from the general case described here.

For the preparation of new material, both approaches offer flexible possibilities for the

course author or programmer. Most teleprocessing systems offer on-line program entry and editing, and some have special-purpose features that actively help the author with questions and prompting. The stand-alone computer terminal can be loaded with a text editor that will enable entering new program material on the tape cassette. In addition, the facility normally used for collecting student responses will permit running an interactive program that will prompt course authors, screen their entries for errors, and store the screened program material on a cassette tape. Program material can also be prepared on a large time-sharing computer and can be loaded onto the cassette when the stand-alone computer terminal is acting as a conventional terminal device.

An important comparison of the two modes of operation is their cost. In either mode, the cost per unit of work accomplished will be less if the system is used efficiently. With CAI systems a common measurement is the cost per student hour. In a situation where a conventional terminal is in operation on a time-sharing system, the cost per student hour decreases as the total amount of terminal use becomes larger. This is because the fixed rental charge for the equipment is divided among a larger number of student hours. The cost cannot go as low as the cost of the time-sharing services, since computer charges and sometimes communication charges increase in approximately a linear relationship with the amount of time used.

With a stand-alone computer terminal, the equipment rental is the only cost involved. In comparison with typical typewriter or cathode-ray tube display terminals, rental charges for a stand-alone terminal will be two-and-a-half to three times higher. Since there is no hourly charge for remote computer services, the cost per student hour will decrease linearly with increased use of the equipment. As an estimate, assuming computer charges of about $5.00 per terminal hour, the overall cost of operating a stand-alone terminal will be about the same as the cost of operating a conventional time-sharing terminal when the total amount of use is between 30 and 40 hr per month. With a greater amount of use, the stand-alone computer terminal will be less costly than the conventional terminal, and a cost of $1.25 per student hour would be possible if the machine could be scheduled for use 200 hr per month.

Typical response times for teleprocessing systems are in the range of $\frac{1}{2}$–5 sec, measured from the time the student response is terminated to the time the computer begins to display the next line of text. The response time varies depending on the number of terminals that are being serviced and on the load from other systems that might be running on the same machine. Modern minicomputers are extremely fast, and in the stand-alone computer terminal all of the computing resources are devoted to a single user. Response to student input is essentially instantaneous, as long as the portion of the program being referenced is currently in the main storage area. If some of the program material must be read in from a tape, the response is delayed and may take several seconds. Appropriate placement of the tape control statements in the program can cause new material to be read from the tape while the student is studying a text display that has just been presented to him.

Since the stand-alone computer terminal does not utilize a communications system in order to operate, its reliability is increased and its cost is decreased. Local, low-speed communications via telephone and acoustic coupler are not expensive, but they are often a source of errors and interrupted terminal sessions. High speed data communication links are expensive and are also a source of potential transmission errors and breakdown. The stand-alone computer terminal does not need to be connected to a time-sharing system for normal operation, and is thus not subject to these problems.

Most users of computerized instruction have a need to collect data on student performance and on the performance of the instructional programs themselves. Although it is not always true, most computer-assisted instructional systems that are available on large time-sharing computers provide a means of data collection. On the stand-alone computer terminal equipped with two built-in tape cassette drives, one tape can be used to hold the instructional program and the other can be used to collect data and student responses under program control. When enough data has been collected, the stand-alone device can access a teleprocessing system via telephone and transmit the data to be stored or processed through statistical analysis.

The major limitation of the stand-alone computer terminal as it is presently configured is its program capacity. In a machine with a core storage of 8000 bytes, about half of the storage is taken up by the system and the remaining 4000 bytes is available for a section of the instructional program. This is a limitation only on the amount of program material that is immediately available, since up to about 100 000 bytes (characters) of program material can be stored on one of the magnetic tape cassettes, and is available with variable amounts of delay depending on its location. At somewhat increased cost, up to 48 000 bytes of memory is available, and this would greatly increase the amount of program material that is available for immediate access. In the near future, it seems likely that this type of machine will regularly be equipped with low-cost disk storage units offering much greater amounts of rapid-access program storage.

The present configuration will hold a useful number of instructional frames in core storage, and will read in additional material from the tape when it is needed. This is entirely adequate for presenting the multiple-choice instructional sequences and self-evaluation programs that are most commonly used in the health sciences as well as other areas. Programs which allow natural language responses from the student can also be accommodated. Programs which simulate clinical interview situations, allowing the student to enter a wide range of questions in his own words, require a very large search algorithm and cannot be run on the stand-alone terminal in its present configuration. Machines available in the near future will have random-access disk storage capabilities that will make it possible to run even these complicated programs on the stand-alone computer terminal.

OTHER APPLICATIONS

In addition to computer-assisted instruction, the stand-alone terminal with two tape cassette units is well suited for a variety of data entry procedures. In the health sciences, the task of collecting initial data on new patients seen in clinics is a possible application. One tape unit can be used to provide a conversational prompting program which will request the needed information. The program can check the data to make sure each entry is within reasonable limits. The second tape unit can collect each data item after it has been checked, and at a later time the data can be transmitted to a large computer for whatever processing or printing may be desired. A connection to a time-sharing system would be required only during the brief time of data transmission.

SUMMARY

Historically, computer-assisted instruction began as a system using a dedicated computer, but most of its development and current use has been on time-sharing computer systems.

Advances in computer technology may result in the stand-alone computer terminal being considered as a viable alternative way of providing CAI capability. A system called PILOT 73 has been written to run on a stand-alone device with built-in minicomputer, character display screen, keyboard and two magnetic tape cassette drives.

General comparisons can be made between the stand-alone computer terminal and the time-sharing system as providers of CAI. The dedicated stand-alone device is likely to provide faster responses and be less susceptible to software and communications problems. The time-sharing system has more capacity to run instructional programs that require a large data base or extensive calculations. Both systems have reasonable facilities for entry of program material, although the time-sharing system can offer more powerful editing and prompting features. In normal use, the stand-alone computer terminal is likely to be less costly per student hour, and this difference will be more pronounced as the amount of use increases.

REFERENCES

1. J. A. STARKWEATHER, Computest: a computer language for individualized testing, instruction, and interviewing, *Psychol. Rep.* **17**, 227 (1965).
2. PILOT 1.6 Guide, Office of Information Systems Report UCS 03.01.01, University of California, San Francisco (1973).
3. J. A. STARKWEATHER, A common language for a variety of conversational programming needs, *Readings in Computer-assisted Instruction*, edited by H. A. WILSON and R. C. ATKINSON, p. 269. Academic Press, New York (1969).
4. J. A. STARKWEATHER, M. KAMP and A. MONTO, Psychiatric interview simulation by computer, *Meth. Inform. Med.* **6**, 15 (1967).
5. M. KAMP, Evaluating the operation of interactive free-response computer programs, *J. Biomed. Systems* **2**, 33 (1971).

PILOT: A Programming Language for Beginners

PILOT is a textually-based computer language providing an alternative to algebraic languages such as BASIC or FORTRAN. PILOT is composed of powerful and nearly syntax-free conversation-processing instructions. Its clarity and simplicity have attracted many teachers, students and adults. Originally designed by John Starkweather as an author language for computer-aided-instruction, it is also an excellent language for initiating beginners into computer programming.

In recent years the language of choice for teaching beginners the art of computer programming has typically been BASIC. BASIC is interactive, it is interpretive (errors are caught as the code is being typed in), and it is widely available. At the Lawrence Hall of Science at the University of California in Berkeley, BASIC has been used successfully with hundreds of children and adults. Most students acquire rudimentary skills in BASIC programming fairly quickly but enough have difficulty with the most fundamental algorithms to warrant pause. These students typically lack a strong background in mathematics, and consequently, the inevitable algebraic explanations become exercises in futility. Though the ratio of the "quick" to the "slow" learners of BASIC varies from place to place, one observation is constant — many people initially interested in learning to program are easily frustrated by algebraic languages. There is no reason why such a large number of people be written-off as "unprepared". There needs to be a better way to initiate the unwary into computer programming. There is. The language is called PILOT.

Why PILOT? The many teachers, students and parents who have already mastered PILOT have little cause to wonder. The language is both easy to learn and use.

Most people remember the awesome effort expended learning (and not learning) English grammar. PILOT users appreciate its scarcity of the computer form of grammer, syntax. Furthermore, PILOT has made both input and output exceptionally straightforward. For example, the code which follows instructs the computer to type the message, "DO YOU KNOW MY NAME", and then to wait for an answer. One line takes care of output, T: (type)

```
T: DO YOU KNOW MY NAME
A:
```

The other handles input, A: (accept an answer). This simplicity characterizes the whole language. Notice, for example, that it is not even necessary to specify a variable for accepting input.

How often can you attain a finished product soon after the idea strikes? PILOT students commonly have that experience. People using PILOT have been able to construct and test their own interesting interactive programs by the first or second session of learning. The power of a textually-oriented, conversational language can be appreciated very quickly, even by people without prior programming experience. At the heart of the language is a feature which makes processing conversation so easy — the "matching" facility.

Through "matching", PILOT can efficiently scan a line of input (in response to a question, for example) for specific words or groups of words. Of the words or sets of words that PILOT attempts to scan for, if but one appears in the input line, the "match" is successful. Any resulting action from the PILOT program may depend upon the success or failure of the match. The example which follows demonstrates "matching".

PILOT types a question	HOW ARE YOU TODAY?
Before retrieving the reply, PILOT is ready to "match" these words with the input	(FINE, GOOD, GREAT, OK, FANTASTIC, HAPPY, WONDERFUL, ALRIGHT)
The reply is typed in	I'M FEELING FINE TODAY
PILOT types its response after scanning the input	GLAD TO HEAR THAT

If the reply were negative, the program would type a different response.

```
HOW ARE YOU TODAY?
?lousy
CAN I DO ANYTHING TO
HELP?
```

Following is a listing of the PILOT program.

```
T: HOW ARE YOU TODAY
A:
M: FINE,GOOD,GREAT,OK,FANTASTIC,HAPPY,
   WONDERFUL,ALRIGHT
```

by Rita May Liff and Keith Vann * See Updates below

Rita May Liff has a B.A. in Mathematics and an M.A. in Education from U.C. Berkeley. She is working on an M.S. in Computer Science at U.C. Berkeley. She has developed curriculum, taught and did teacher training in computer and mathematics courses for the Computer Education Project at Lawrence Hall of Science, U.C. Berkeley and has taught computer science and math at Mills College, Oakland, California. She is now a software engineer at Zilog, Inc., Cupertino, California.

Keith Vann is an undergraduate in Computer Science at U.C. Berkely. He developed extensive curriculum for the teaching of PILOT for the Computer Education Project at Lawrence Hall of Science and is the author of a PILOT reference manual in use at Lawrence Hall.

(This article which appeared in the September 1978 issue of INTERFACE AGE Magazine, pp. 64-67 is reprinted by permission of INTERFACE AGE Magazine, 16704 Marquardt Avenue, Cerritos, California 90701)

TY: GLAD TO HEAR THAT
TN: CAN I DO ANYTHING TO HELP?

- **T:** tells the computer to *type* a message
- **A:** instructs the computer to *accept an answer*
- **M:** holds the words to be *"matched"* (scanned for)
- **TY:** *types* its message *if yes*, the match was successful
- **TN:** *types if no*, the match was unsuccessful

Core PILOT also includes instructions for subroutines and computations, as well as two types of variables. The core language is summarized briefly here. Many versions of PILOT include a wide variety of extensions as well.

THE EIGHT CORE PILOT INSTRUCTIONS

- **T:** type a message
- **A:** accept an answer
- **M:** match for keywords
- **J:** jump to labeled line
- **U:** use a procedure
- **R:** remarks-from the programmer
- **C:** compute a result
- **E:** end procedure

TWO PILOT CONDITIONERS

- **Y:** make instruction conditional upon a positive match
- **N:** make instruction conditional upon no match

TWO TYPES OF PILOT VARIABLES

$STRING — for text

#NUMERIC — for computations

PILOT IN THE CLASSROOM

Children and PILOT seem to mix particularly well. PILOT instructions are simple and clear enough to facilitate introducing programming concepts such as: input and output, transfer-of-control, looping, and branching. Consider this program written by a third-grade student from Anna Yates School, in Emeryville, California.

SAMPLE RUN:

```
WHAT IS ROUND
?BALL
AND BRIGHT
?SUN
AND ON THE CEILING
?LIGHT BULB
YOU GOT IT!

*FIN!
```

ANOTHER RUN:

```
WHAT IS ROUND
?MOON
AND BRIGHT
?STARS
AND ON THE CEILING
?HANGING LAMP
SORRY, I WAS THINKING OF LIGHT BULB.

*FIN!
```

PROGRAM LISTING:

```
          T:WHAT IS ROUND
          A:
          M:LIGHT,BULB
          JY:*RIGHTON
          T:AND BRIGHT
          A:
          M:LIGHT,BULB
          JY:*RIGHTON
          T:AND ON THE CEILING
          A:
          M:LIGHT,BULB
*RIGHTON  TY:YOU GOT IT!
          TN:SORRY, I WAS THINKING OF LIGHT BULB.
          E:
```

This riddle program requires branching whenever a correct answer is input so that no further clues are given. The use of the JY: (jump if yes) instruction handles the situation nicely and is easy to explain to students.

It is important to consider introducing young children to programming for a variety of reasons. First of all, despite the simplicity of PILOT, the constructs learned will apply to any future programming experience, regardless of how symbolic or abstract the languages used later are. Also, the process of "teaching" the computer to carry on a conversation serves to dispel the myth of a computer's intelligence; students soon realize that it is the programmer who imparts the "wisdom" (or lack of it). This realization is not likely to occur to the same degree through mere interaction with programs written by others. In addition, programming provides a medium for experimentation with different organizational models. Flow-charts and tree-diagrams, and ultimately the programs themselves, represent ways to visualize decision-making processes.

The affective component of education should not be ignored either. Perhaps most notable about the program just shown was the surge of pride in its designer when she sees others enjoying her interactive riddle. This sense of accomplishment is fundamental to learning. Because an interactive program represents a kind of 'game' that the author's peers may want to play, a great deal of motivation is provided. Students work hard at creating the questions and remarks which will be displayed.

The program which follows was designed by three fifth-grade students from Sun Valley School in San Rafael, California. The instructions used are no more complex than those needed to create the simple riddle program. But these students incorporated information they were learning outside of their PILOT class into their program. The students had been studying 'pond ecology' and each group investigated a particular organism. Their PILOT programs were designed to continue giving clues about a 'mystery organism' until a correct response is input or until the clues run out.

THE LISTING:

```
R:WRITTEN BY FRANCESCA BERTONE, KATHY ROBINSON, JULIE LEET
R:SUN VALLEY SCHOOL, 5TH GRADE
R:MAY 13TH, 1974
T:
T:WE'RE GOING TO PLAY A QUESTION AND ANSWER GAME
T:DO YOU WANT TO PLAY
A:
M:YES,YEAH,OK,O.K.,SURE,FINE,ALRIGHT
TY:HERE'S YOUR FIRST CLUE:
JN:*QUIT
T:TAKE A GUESS.  I AM AN ANIMAL
A:
M:SNAKE
TY:WOW!!!!!!
JY:*END
```

```
      T:I AM A VERTABRATE
      A:
      M:SNAKE
      TY:GREAT!!!!!
      JY:*END
      T:MY LATIN NAME IS THAMNOPHIS
      A:
      M:SNAKE
      TY:FAR OUT!!!!
      JY:*END
      T:I EAT AMPHIBIANS AND INSECTS
      A:
      M:SNAKE
      TY:GOSH, YOU'RE SMART
      JY:*END
      T:I EAT MY FOOD ALIVE
      A:
      M:SNAKE
      TY:CONGRATULATIONS!!!!
      JY:*END
      T:I DO NOT LIVE IN THE POND
      A:
      M:SNAKE
      TY:BY GOLLY, I NEVER THOUGHT YOU'D GET IT SO SOON!!!!
      JY:*END
      T:I GROW TO A MAXIMUM LENGTH OF THREE FEET
      A:
      M:SNAKE
      TY:THRILLS, YOU GOT IT!!!!!
      JY:*END
      T:AREN'T YOU EVER GOING TO GET IT
      T:I'M RELATED TO THE PTERODACTYL
      A:
      M:SNAKE
      TY:I'M AMAZED!!!!!!!!
      JY:*END
      T:I BEAR 50 OR MORE LIVE YOUNG IN THE SPRING
      A:
      M:SNAKE
      TY:GOOD!!!
      JY:*END
      T:I EMIT A FOUL SMELL WHEN SCARED
      A:
      M:SNAKE
      TY:YOU GOT IT
      JY:*END
      T:I DO NOT HAVE EARS
      A:
      M:SNAKE
      TY:HOW DID YOU KNOW???????
      JY:*END
      T:I HAVE A YELLOW STRIPE DOWN MY BACK
      A:
      M:SNAKE
      TY:EXCELLENT!!!!
      JY:*END
      T:MY BODY IS COVERED WITH SCALES
      A:
      M:SNAKE
      TY:YOU'RE SMART
      JY:*END
      T:I AM COLDBLOODED
      A:
      M:SNAKE
      TY:PRETTY GOOD!!!
      JY:*END
      T:I AM A REPTILE
      A:
      M:SNAKE
      TY:FINALLY!!!!!!!
*END  T: HERE'S A CRUDE PICTURE OF ME
      T:
      T:                                    XXXXXXXX
      T:                                  XX       X
      T:         XXXXXXXXXX             XX     XXXX   X
      T:       XX          XX          X     XX     XXXX
      T:     X   XXXXXXXX   XXXXXXXXX   XX           X
      T:    X  XX          X             XX
      T:   X XX            XXX         XX
      T:  XXX                XXXXXXX
      T:XX
      T:
      T:                  GARTER  SNAKE
      T:
      T:
      E:
*QUIT T:WELL IF YOU DIDN'T WANT TO PLAY THE GAME
      T:WHY DID YOU ASK FOR THE PROGRAM??
```

SAMPLE RUN:

```
WE'RE GOING TO PLAY A QUESTION AND ANSWER GAME
DO YOU WANT TO PLAY
?YES
HERE'S YOUR FIRST CLUE:
TAKE A GUESS.  I AM AN ANIMAL
?DOG
I AM A VERTABRATE
?FISH
MY LATIN NAME IS THAMNOPHIS
?FROG
I EAT AMPHIBIANS AND INSECTS
?TURTLE
I EAT MY FOOD ALIVE
?LIZARD
I DO NOT LIVE IN THE POND
?RACCOON
I GROW TO A MAXIMUM LENGTH OF THREE FEET
?A CAT
AREN'T YOU EVER GOING TO GET IT
I'M RELATED TO THE PTERODACTYL
?LIZARD
I BEAR 50 OR MORE LIVE YOUNG IN THE SPRING
?I DON'T KNOW
I EMIT A FOUL SMELL WHEN SCARED
?SKUNK
I DO NOT HAVE EARS
?A SNAKE
HOW DID YOU KNOW???????
 HERE'S A CRUDE PICTURE OF ME
                                    XXXXXXXX
                                  XX       X
        XXXXXXXXXX              XX     XXXX   X
      XX          XX          X     XX     XXXX
    X   XXXXXXXX   XXXXXXXXX   XX           X
   X  XX          X             XX
  X XX            XXX         XX
 XXX                XXXXXXX
XX

                  GARTER  SNAKE
*FINI
```

After writing several of the questions and responses in a program such as this one, students on their own often notice the repetition in code they are generating. When they ask if there is any way to get around writing in this way, it is clear that they are "ready" to be taught about subroutines and other programming mechanisms that will be helpful. In this context, motivation for introducing new concepts has been established.

PEOPLE PRAISE PILOT

"PILOT provides a language which is simple almost to the point of being uninteresting to some computer professionals. But it allows a person with no prior computer experience to interact with a machine in a human way. It provides a newcomer with immediate reinforcement of success, and graphically demonstrates some of the more obvious features of what a computer is like." So says Dr. John Starkweather, author and developer of PILOT. "When I began thinking about a language like PILOT, there was no language oriented toward natural language and conversational interaction. I tried to develop simple mechanisms that would be easily understandable."

PILOT has been appraised as an easy-to-learn language by numerous others who consider it ideal for introducing programming constructs to the novice. At Lawrence Hall Pete Rowe who originally implemented it there remarks, "I could see that some kids had the ability to program, but not with BASIC. There are people who are more word-oriented than numerically-oriented, and I felt they would respond better to PILOT." Joyce Hakansson, coordinator of computer classes there, believes that "there is an organizational skill involved in creating computer programs. . .PILOT gives beginning computer users access to that kind of creation much faster than more traditional languages."

Dean Brown, a long-time advocate of PILOT, has implemented PILOT on a Z-80 and has been responsible for its disemmination internationally. Dean tells us, "PILOT programs have been written in Yugoslavian, French, Spanish and Arabic. . .with support from UNESCO and the World Bank." He feels that "PILOT is so general and

teaching." He notes that when a computer is in a dialog with a student, they are exchanging word strings. "There are only several things that can happen, and they are easy to do with PILOT. You can type in a word string (T:), you can get one back (A:), you can interpret a word string (M:) and branch on content (TY: and TN:). You can also imbed earlier responses in current strings. And PILOT can be learned in a hurry."

Greg Yob has also been influential in spreading the good work about PILOT. In 1975 he formed the PILOT Information Exchange, a clearinghouse for information and resources on PILOT. Greg feels that, "PILOT is likely to become the de facto home computer education language. It's easy to learn, not hard to implement and void of functions unneeded in home education. It's important to have a system not cluttered by things a home user would not be interested in."

Earl Keyser has recently taken over the Exchange. He is implementing PILOT on an Apple II and likes PILOT because "in 8 commands it can do what most author languages do at best poorly and with many more commands . . .It is elegant but not simple-minded." He has found PILOT the preferred language for training teachers to teach their own students computer programming.

A completely supported microcomputer implementation of PILOT is now available from Processor Technology. Implemented by John Starkweather, it runs on a SOL and may be obtained by writing to:

Processor Technology
PILOT 8080, Version 2.2
7100 Johnson Industrial Drive
Pleasanton, CA 94566
(415) 829-2600

Dean Brown may be contacted for listings of his Z-80 version of PILOT, which runs on an MCZ system. He will also send along sample PILOT programs, some designed for children and others developed in a Creative Writing Course at Stanford. Send inquiries to:

Dean Brown
Zilog, Inc.
10460 Bubb Road
Cupertino, CA 95014
(408) 446-4666

Many other implementations exist, and Earl Keyser has the latest information through the PILOT Information Exchange. Join the Exchange or write to him for resources:

Earl Keyser
22 Clover Lane
Mason City, Iowa 50401
(515) 424-5548

Other References on Educational Uses of PILOT

Dean Brown, et al. A Pilot Experiment in Educational Technology Using Computers in the Affective Domain. SRI Project 072531-017 (Menlo Park: SRI), Oct.,1969.

Dean Brown & Mohammed El-Ghanam. Computers for Teaching, Computer, ~~March~~ January 1975, pp. 16-22.

Ellen Nold & Sallie Cannom. PILOT (a series of three articles). People's Computers, July/Aug, Sept/Oct, & Nov/Dec, 1977 (available from PCC, Box E, Menlo Park, Calif. 94025). Describes PILOT programs used in teaching Freshman English courses at Stanford.

Charles Shapiro. A BASIC PILOT. People's Computers, Sept/Oct, 1977. Describes a BASIC interpreter for PILOT, written by a high school student.

John Starkweather, A common language for conversational programming needs. in H. A. Wilson & R. C. Atkinson, Computer Assisted Instruction- A Book of Readings (NY: Academic Press), 1969. Describes the first implementation of PILOT and design rationale, by the original author of the language.

John Starkweather. Guide to 8080 PILOT, Version 1.1. Dr. Dobbs Journal, April, 1977, 17-29 (available from PCC at address above). Also available from Natl. Library of Medicine, Center for Biomedical Communications, Bethesda, Md. This describes the first implementation of PILOT for a microprocessor (8080).

Greg Yob. PILOT. Creative Computing, May/June, 1977, pp. 57-63. Another summary of core PILOT for the beginning user, written by one of its strongest proponents.

People Resources

The following are updated addresses/locations of key people who have used PILOT in educational settings (revised from attached article).

Rita Liff Levinson (nee Rita May Liff) received her M.S. in Computer Science from U. C., Berkeley, and is now at TAK Components, 1307 N. Carolan Ave., Burlingame, California 94010.

Keith Vann ~~is now finishing~~ has recently finished an M.A. in Computer & Information Sciences at the University of Califonnia at Santa Cruz, and is now working for Hewlett-Packard Corporation.

Dean Brown now has his own company, called Picodyne, Portola Valley, California.

Phyllis Cole (who edited People's Computers)is now at Apple Computer, Inc., in Cupertino, Calif.

Dr. John Starkweather, originator of PILOT, is now at Langley Porter Institute, University of California, San Francisco, California.

Pete Rowe now has his own company ~~and may be reached c/o Lawrence Hall of Science, University of California~~ for developing innovative CAI in, Berkeley, Calif. ~~94720~~.

Computers and Society

David D. Thornburg
Innovision
P.O. Box 1317, Los Altos, CA 94022

As R. Buckminster Fuller is fond of pointing out, synergy is the behavior of whole systems which is not predicted by the behavior of the parts taken separately. There have been two recent developments in the personal computer world which, taken together, have the promise of true synergism. These events are the publication of Seymour Papert's long-awaited book: **Mindstorms - Children, Computers, and Powerful Ideas** (Basic Books), and the introduction of the language PILOT for the Atari computers.

Since this year's theme is communications, it is only appropriate that we spend some time looking at the communication between the user and the computer. While the mechanical means through which this communication takes place are worthy of extensive discussion, I want to concentrate this month on the nature of the language through which we interact with computers, since this also is an area of intense importance.

One might argue that there is little reason for concern with computer languages at this point, since we all have access to fairly powerful versions of BASIC on our computers. We might all agree that BASIC is not terribly hard to learn, and that there are lots of very exciting BASIC programs, and even that BASIC has become the *de facto* standard computer language for personal computers.

But even with the tremendous penetration of BASIC in the marketplace, I have yet to find any serious computer user (regardless of age) who really likes it. At the primitive level of programming at which we all start, BASIC works pretty well. But as we get more sophisticated, most of us find ourselves writing code that we can't understand two weeks after we write it.

Of course, there are detractors of BASIC who feel that languages like PASCAL are the solution. I must confess that I find PASCAL lacking in that it doesn't encourage the user to sit down and start writing some small part of a program, to play with the bits and pieces, and to then bring everything together later on. This is one area in which BASIC excels. For those who feel that people should be organized when they write a program, PASCAL (and C and other "serious" languages) may very well be the best choice. But what about the new computer user who wants to build a highly interactive program, or the child who wants to explore concepts in geometry through the experience of programming rather than through playing a pre-defined "game" or sitting at a "canned" CAI lesson? These people need languages which are interactive, highly flexible, extraordinarily powerful, and are easy to get started with.

LOGO And PILOT Are Two Such Languages. While LOGO (as this is being written) does not yet exist on commercial personal computers, it has been the subject of an extensive research program at MIT for more than a decade. Much of the research has been devoted less to the development of computer languages *per se*, than to the development of a computer assisted learning environment for children. The goals, aspirations and results of this work are the subject of Papert's **Mindstorms.** It is hard to imagine any person who is intensely concerned with the use of computers by children who would fail to be moved by the sweeping vision implicit in Papert's work. Writing from the perspective of a mathemetician who spent much time with Jean Paiget, Papert presents a variation on the Paigetian model of the "child as builder" in that he sees the need for children to have an abundance of materials with which to build things.

...for some educators, Computer Aided Instruction has come to mean "computers programming children".

That the computer can be one such building tool is the cornerstone of the many computer literacy activities we see springing up around the world. But, for some educators, Computer Aided Instruction (CAI) has come to mean "computers programming children". There is much to be gained from reversing this process — and that is where the need arises for an exceptionally powerful (and easy to learn) computer language.

LOGO is a highly interactive language which contains a graphics environment containing something called a "Turtle". The Turtle is a non-Euclidian point (having both position and direction, rather than a position alone). The programmer can send messages to the Turtle which cause pictures to be drawn on the display screen. Those of you who are familiar with the Milton Bradley **Big Trak** are already familiar with this idea. To draw a square on the screen, for example, a child working in the Turtle microworld might type:

```
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
```

As each instruction is executed, the Turtle first moves forward by 100 units, and then turns right by 90 degrees, drawing its path on the screen as it moves. The desired square thus takes shape on the screen. A programmer wishing to use squares quite frequently might wish to extend the repetoire of commands the Turtle uses by defining a new procedure which the Turtle then "understands":

```
TO SQUARE
REPEAT 4
  FORWARD 100
  RIGHT 90
END
```

If squares of arbitrary size are required, one might write:

```
TO SQUARE :SIZE
REPEAT 4
  FORWARD :SIZE
  RIGHT 90
END
```

and then, anytime a square is desired, one would type, for example,

```
SQUARE 47
```

to draw a square with each side 47 units long.

The value of using the Turtle environment in an interactive way is expressed by Papert this way:

> *Working in Turtle microworlds is a model for what it is to get to know an idea the way you know a person. Students who work in these environments certainly do discover facts, make propositional generalizations, and learn skills. But the primary learning experience is not one of memorizing facts or of practicing skills. Rather, it is getting to know the Turtle, exploring what a Turtle can and cannot do. It is similar to the child's every day activities, such as making mudpies and testing the limits of parental authority — all of which have a component of "getting to know".*

One of the more valuable experiences for children involved with computers is learning how to "debug" a program with errors in it. Traditionally, we are taught that errors are bad. Papert shows that, by accepting the inevitability of errors in programs, children can learn to analyze the results of the error and then learn to avoid the error in the future, and to "patch" it in the meantime.

In order to make the debugging process as meaningful as possible, children are asked to find the bug by "playing Turtle". The child then walks

...children are asked to find the bug by "playing Turtle".

around the floor, executing each instruction in turn until the "faulty" instruction is found. But doesn't this method for finding errors lead the child to "thinking like the computer"? Papert sees this experience in a larger context. He says:

> *In my experience, the fact that I ask myself to "think like a computer" does not close off other epistemologies. It simply opens new ways for approaching thinking. The cultural assimilation of the computer presence will give rise to computer literacy. This phrase is often taken as meaning knowing how to program, or knowing about the varied uses made of computers. But true computer literacy is not just knowing how to make use of computers and computational ideas. It is knowing when it is appropriate to do so.*

While Papert is quite heartened by the growth of the personal computer industry, since this growth will result in children having ever easier access to computers, he is frustrated by the limitations of these machines and by the extremely strong penetration of BASIC into the marketplace. He likens BASIC to the QWERTY layout on the keyboard — an artifact from a time when better ways didn't exist. But what of LOGO itself? This language will not be forever locked in the University laboratory. Versions for the TI 99/4 and the Apple computer will probably come into general availability soon.

He likens BASIC to the QWERTY layout on the keyboard—an artifact from a time when better things didn't exist.

Even if LOGO, with all its power, doesn't make its appearance in the marketplace soon, I feel that many of Papert's ideas can be implemented today on the small computers whose capabilities he dislikes, through the medium of the language PILOT.

As normally written, PILOT interpreters allow the user to create spectacular text manipulation programs (*c.f.*, the article by Thornburg and Thornburg in the first issue of **COMPUTE!**). Recent embellishments have made PILOT a splendid language to use on computers with high quality graphics environments, such as the Atari 400 and 800. Those of us who use Atari computers can, with Atari PILOT, do many of the things Papert does with LOGO.

Those of you who are familiar with PILOT probably think of it as a language best suited for creating text-based learning materials. My view of the language is far more open than that, because it is so easy to teach to youngsters. It has long been my dream to see the superb text manipulative power of PILOT extended to give the user similar power to

create pictures. The Atari PILOT is the answer to this dream since it contains a graphics package that is, in some ways, very similar to the Turtle graphics of LOGO.

For example, an Atari computer user running PILOT might draw a square this way:

```
GR: CLEAR
GR: DRAW 25
GR: TURN 90
GR: DRAW 25
GR: TURN 90
GR: DRAW 25
GR: TURN 90
GR: DRAW 25
GR: TURN 90
```

As each instruction is carried out, the square begins to take shape on the screen. If the user wants to draw lots of squares PILOT allows one to create a "module" as a deferred program. By typing AUTO at the command level, and then typing:

```
*SQUARE
  GR: 4(DRAW 25; TURN 90)
E:
```

A module (SQUARE) is created. On leaving the AUTO mode (the AUTO mode automatically places line numbers in front of each statement, thus keeping them from being executed immediately), the user can draw a square by typing:

```
U: *SQUARE
```

in which **U:** is the USE operator found in all versions of PILOT.

My reasons for giving this particular example are two-fold. First, it shows the similarity between the Turtle graphics of Atari PILOT and that of LOGO. Secondly, it shows that PILOT can be used in an interactive mode which combines deferred program segments (modules) with immediate execution of commands.

Can (or should) PILOT replace BASIC? I can only answer by saying that, on the Atari computers, it has for me. I find the language much easier to learn, much easier to use, and capable of doing anything I have ever wanted to do. One of its major features (especially important when working with children) is that a PILOT program can be read by someone other than its author. This is rarely the case for large BASIC programs.

Finally, while most users will want to use PILOT to write self contained programs, I am very happy with the fact that the Atari implementation of PILOT allows the user to interact with the language without having to write "finished" programs. As Papert has shown, the value of "playing around" with an interactive language can be great for all users, and especially for children. ©

Editor's Note: Atari PILOT is not expected to be available until late spring. Check with your dealer for more information. RCL

DAVID THORNBURG

PILOT is an exceptional language for text manipulation. One of its major features is the ease with which it allows the programmer to accept and evaluate lengthy responses from the user. To illustrate this, let's look at a short program which tests the user's knowledge of the dietary habits of dragons.

Writing the program ...

To write this program, turn on the computer, type AUTO, and press the RETURN key. The change in color of the screen indicates that you are in the deferred programming mode. Any PILOT commands that are typed will be saved as part of a program.

First let's type the question we want answered. Type:

```
T: CAN YOU NAME SOMETHING WHICH DRAGONS LIKE
TO EAT?
```

When this program is run, this line will type (T:) the question CAN YOU NAME SOMETHING WHICH DRAGONS LIKE TO EAT? on the screen. If you are familiar with BASIC, you will notice some differences between T: and the corresponding command in BASIC, PRINT. First, there are no string delimiters in PILOT. The user doesn't have to enclose text in quotation marks. Another important property of T: is that words will never be broken as they are printed. If a given word will not fit at the end of a line being typed, PILOT places this word at the start of the next line.

Now that we have "asked the question", we need to accept a response from the user. This is done with the PILOT "accept" command, A:. Type:

```
A: $FOOD
```

This command will accept a response from the keyboard and store it in two places. The first place the response will be stored is called the "accept buffer". This is a 256 character buffer which PILOT can "look at" for text analysis. We have also placed the user's response in the string variable $FOOD. String variable names in PILOT always start with the dollar sign ($).

Now that we have a response, we want to test this response. There are lots of things which could be typed in response to the question. We will concentrate on four kinds of answers:

1. The user typed YES, without being specific.

2. The user typed NO, or some other negative response.

3. The user gave a correct response (PEOPLE, etc.)

4. The user gave an incorrect response (SHOELACES, etc.)

To test for specific responses, PILOT has a tremendously powerful command called "match" (M:). This command lets us compare anything in the accept buffer against a list of items to see if any item in that list appears in the user's response. To test for a positive response, for example, just type:

M: YES, YUP, OK, FINE, SURE

When PILOT executes this line, the contents of the accept buffer will be tested to see if any of the items in this list appear in the response. If there is a "hit" - a succesful match - then PILOT will set a "yes flag" to be true, and a "no flag" to be false. These flags have the names Y and N and can be used to make any PILOT command operate conditionally. For example, we have tested our response to see if it was YES (or YUP, etc.). If this match worked, we want to ask the user to give a specific example of dragon food. We need to conditionally type another line of text. To do this just type the following:

TY:PLEASE NAME SOMETHING THEN.

This command will only be executed if the Y flag is "true". Next we need to conditionally accept a new value for $FOOD. Type:

AY: $FOOD

Now that we have tested for a positive response, let's test for a negative response. Type:

M: NO, UNSURE, UNCERTAIN

If this match command worked, we want to stop the program, since the user doesn't have an answer. Keep in mind that if this match command worked, the Y flag will be true. We can then type a message on the screen and jump (J:) to another part of the program. To do this just type:

TY:I'M SORRY TO HEAR THAT.

JY: *FRED

The second line above is a conditional jump command which causes PILOT to branch to the label *FRED. Labels in PILOT can have any name you wish. Since PILOT doesn't use line numbers, these labels give us a way of jumping around from one place to another in the program.

Assuming our program has gotten this far, we are now ready to test to see if the user has entered a correct assessment of the sort of things dragons like to eat. To do this we use the match command again. Type:

M: PEOPLE, PRIN, KNI, NITE, CAKE, PRETZ, FLOWER

This match command will check for words like PEOPLE, and words starting with PRIN (princes, princesses, printers), quite a few spellings of knight, and will check for CAKE and PRETZELS too! Now we are ready to respond to the user with the following two statements. Type:

TY:YES, DRAGONS DO EAT $FOOD.

TN:I DIDN'T KNOW THAT DRAGONS ATE $FOOD.

Since these are both conditional print statements, and each uses a different flag, only one of them will be printed on the screen. As you can see, the response that the user typed will be included in the typed response. When PILOT sees $FOOD while typing the line, it checks to see if there is a string variable with that name. If there is, then the contents of the variable will be typed. If the variable has not been defined, then its name will be typed instead.

We are almost finished with our program. We must now type the ending:

*FRED

T:THAT'S ALL FOR NOW.

E:

The first line above is just the label *FRED. If the previously typed JY: command ever gets used, the program will jump to this label. By now you know

what the T: command does. The E: command is the PILOT "end" command which is used to terminate the program.

Now that you are all done, press the RETURN key again and the screen will revert back to its original blue background. You are now ready to run this program.

Running the program ...

To run this program, just type RUN and press the RETURN key. You should see the question CAN YOU NAME SOMETHING WHICH DRAGONS LIKE TO EAT? displayed at the top of the screen. Notice that the words LIKE TO EAT? appear on the next line of the screen without any of the words being broken at the line end.

Next you should answer this question. I suggest that you type in an answer like this (please don't press RETURN at the end just yet!):

CAN I NAME SOMETHING WHICH DRAGONS LIKE TO EAT? WELL, SURE I CAN!

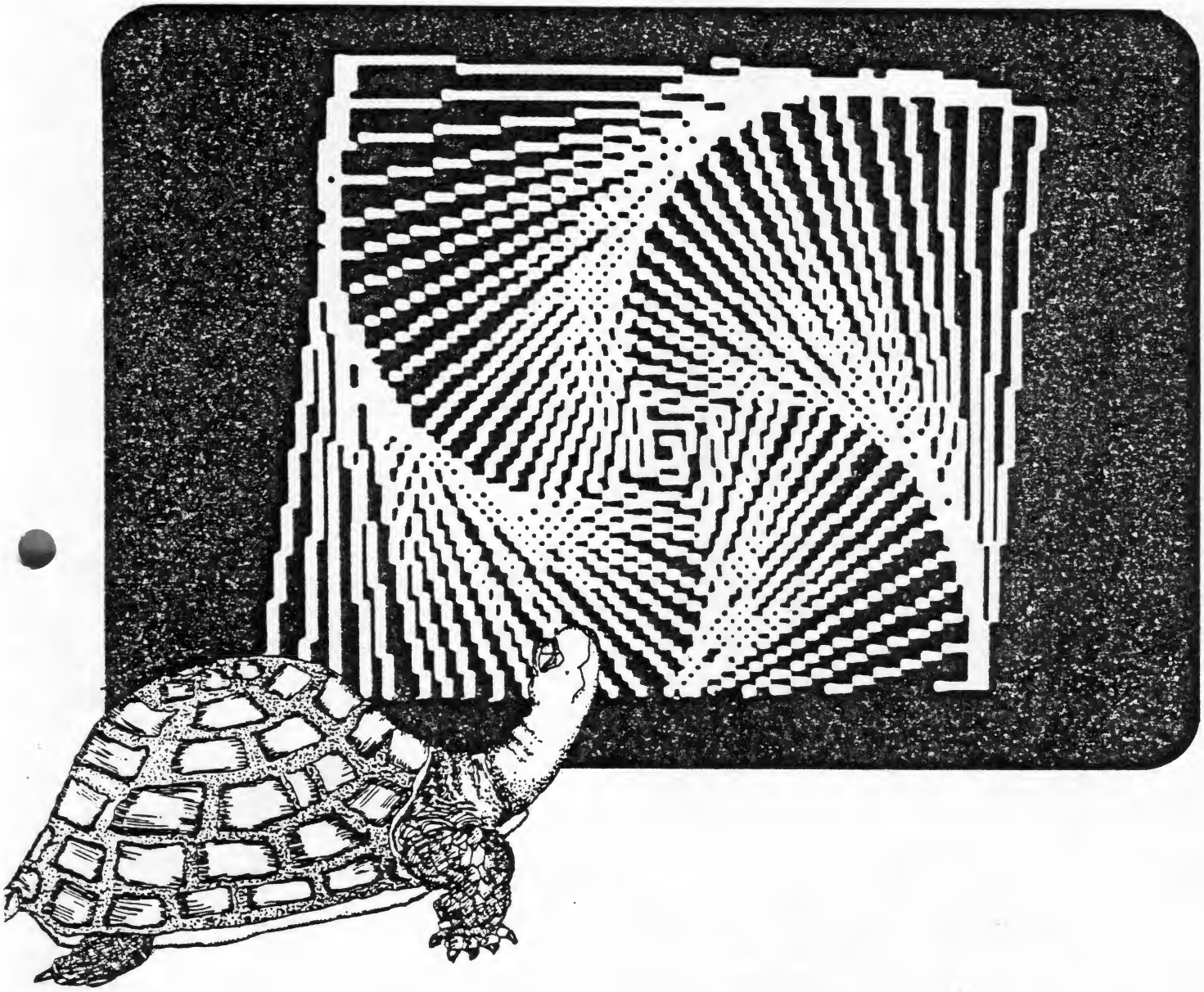
A you can see, this is a lengthy response. Now watch the screen closely as you press the RETURN key. PRESTO! PILOT perfomed the match of YES, YUP, OK, FINE, and SURE against a fairly lengthy response, and it finished the job almost instantaneously. I know of no other interpreted language on an 8-bit computer which can do list processing this fast.

Now answer the next question by typing:

ORANGES AND LIMES

and notice how quickly PILOT gives the final response.

Atari PILOT is one of the fastest running implementations of this language ever written. When this excellent text manipulation capability is considered in light of the exeptional turtle graphics environment contained in this language, it is easy to see why PILOT will be the "Language for People".



Picture This!

PILOT'S Turtle Graphics for Atari

by David D. Thornburg

check this out by "playing turtle" yourself. Try following each of the instructions above while walking around (you might want to take fewer than 25 steps on each side, of course), and see where you end up.

If we now type:

GR: CLEAR

all that happens is that the lines on the display are erased, but the turtle isn't moved at all. If you aren't sure about this, just type:

GR: DRAW 10

and notice that the turtle drew a line to the left, rather than one pointing straight up. The turtle can be placed in his starting position with the commands:

GR: GOTO 0,0; TURNTO 0

This line of instructions contains two *absolute* commands. GOTO picks the turtle up and moves him to a given location on the screen (0,0 is the home location; you should try other combinations such as 7,15 or 12,3 to see what each of the two numbers does). The command TURNTO changes the orientation of the turtle to a specified value (measured in degrees). TURNTO 0 places the turtle facing straight up.

Atari PILOT gives us an even more convenient way to draw figures like squares. First enter:

GR: GOTO 0,0; TURNTO 0; CLEAR

This gets us started at the right place. Now type:

GR: 4(DRAW 25; TURN 90)

Wow! One line of instructions can create a complete figure!

Have you figured out what went on when the last command was entered? This command instructed the turtle to do something four times. The things we wanted the turtle to do were placed inside the parentheses "()". In effect, this command says: "Four times you are to both draw a line 25 units long and then turn right by 90 degrees." Not elegant English, but pretty good Turtletalk, none the less.

To see some more figures, type this:

GR: PEN YELLOW
GR: 5(DRAW 25; TURN 72)

(See Figure 5)

These instructions created a picture of a regular pentagon.

As you can see, the turtle can be made to draw some very nice figures; but there is even more he can do.

Atari PILOT allows the user to build a dictionary of procedures (called *modules*) which the turtle can use. These modules are "saved" to be used when needed, rather than being used as they are being written. To create a module, first leave the graphics mode by typing:

GR: QUIT

and then type AUTO. As soon as you press RETURN after typing AUTO, the screen changes from blue to yellow. If you are using a black and white display, the AUTO mode will show dark letters on a light background. This color change is PILOT's way of indicating that each line of commands you enter is going to be saved for use later, rather than being used right away. The way that Atari PILOT distinguishes between immediately executed instructions (like those we have been using thus far) and instructions to be executed later (like those we are going to use to create dictionary entries) is by placing line numbers in front of all deferred instructions – just as in BASIC. Since PILOT doesn't use these numbers for anything else (*unlike* BASIC), the user doesn't see the numbers as lines are being entered in the AUTO mode.

Modules have three parts – a name (called a *label*), the instructions the module is to perform, and an *end* command. Here is a simple example to demonstrate how modules work. First, type the name:

*STAR

All labels and names of modules start with an asterisk (*). Next, type the recipe for a star (trust me):

GR: 5(DRAW 35; TURN 144)

and, finally, finish the module with the *end* command:

E:

When these commands are entered, the screen shows a square spiral growing out from the center. The reason the squiral keeps growing is that each time the jump (J:) command is used, the length of the next side (#A) is increased by one screen unit and a new side is drawn. To stop this program (we have put it into an endless loop), just press the break key on the computer.

By making a few changes to *SQUIRAL, we can have the computer create lots of pretty figures. First type:

GR: QUIT

and

LIST

Here is what you should see on the screen:

```
10 *SQUIRAL
20 GR: GOTO 0,0; TURNTO 0; CLEAR
30 C: #A=0
40 *DRAWLINE
50 C: #A=#A+1
60 GR: DRAW #A; TURN 90
70 J: *DRAWLINE
80 E:
```

What we are going to do next is modify *SQUIRAL to allow different angles to be chosen when drawing each figure. This means that we will have to replace the fixed value of 90 degrees (see line 60 in the listing) with another number (#B) which will be entered from the keyboard. Here is how to do this. Type:

```
12 T: PLEASE ENTER AN ANGLE
14 A: #B
60 GR: DRAW #A; TURN #B
        [this replaces our old line 60]
```

Next, we should make the module stop itself when the picture gets too large, so we don't have to press the BREAK key. To do this we need to make the *jump* command in line 70 operate only on the condition that the picture isn't too big. Let's say that the picture should keep growing as long as the length of the last side (#A) is less than 100 units long. We can do this by changing line 70 this way:

70 J (#A<100): *DRAWLINE

Finally, it would be nice to have *SQUIRAL start all over again when a picture is finished. To do this type the following line:

75 J: *SQUIRAL

Now we are ready for some more pretty pictures. Enter:

U: *SQUIRAL

Now, instead of drawing anything, there is a message in the blue area which says

PLEASE ENTER AN ANGLE

Just for fun, enter 91 and press RETURN

(See Figure 9)

When the longest side of this squiral equals 99 screen units, the module will stop drawing the figure and the display will ask for a new angle to be entered. The module can be stopped at any time by pressing the BREAK or the SYSTEM RESET key.

It is impossible in this short space to do more than hint at the utility of the graphics on Atari PILOT. I hope that this brief tour of PILOT has demonstrated its potential for displacing BASIC as the first language for neophyte programmers. ■

FOOTNOTES

1 Seymour Papert, *Mindstorms – Children, Computers and Powerful Ideas*, Basic Books, 1980.

2 Alan Kay, "Microelectronics and the Personal Computer," *Scientific American*, September 1977, pp. 230-244.

3 Li-Chen Wang, "An Interactive Programming Language for Control of Robots," *Dr. Dobb's Journal*, V. 2, No. 10, November 1977, p. 10, ff. ago).

4 Big Trak was reviewed in the July-August 1980 issue of *Recreational Computing*.

Figure 9